Préambule :

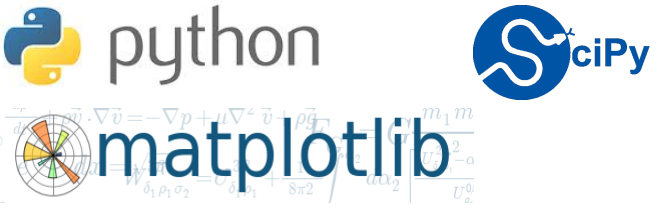
Il n'est pas d'usage de systématiquement résoudre de manière analytique (formelle) les équations différentielles que l'on rencontre dans le domaine scientifique, cela pour deux raisons :

- *la diversité des équations et des méthodes qui leur sont adaptées rendraient ce travail fastidieux,*
- *de très nombreuses équations différentielles sont sans solution analytique (!).*

On fait généralement appel à des programmes (algorithmes) qui recherchent une solution numérique approchée de la solution selon différentes méthodes dites d' "intégration numérique".

Objectif :

L'objectif de ce TD est de mettre en œuvre et de comparer les performances de méthodes d'intégration numérique par approximation d'équations différentielles d'ordre 1, la méthode mise en œuvre sera celle d'Euler avec un schéma explicite et elle sera comparée à la méthode optimisée "odeint" de la bibliothèque "scipy.integrate".



A – Rappel sur la notion de "schéma" d'intégration ou de dérivation numérique.

Dans de très nombreux cas, on est amené à devoir estimer la valeur de la pente d'une fonction en un point donné sans connaissance de la fonction exacte, mais uniquement à partir d'une série de valeurs approchées. Soit $x(t)$ *, une fonction dont on connaît des valeurs pour des dates* $(t_0,\ldots,t_{N-1},t_N)$ *.*

La figure et les relations ci-dessous permettent d'illustrer les différents schémas d'estimation de la dérivée et le vocabulaire associé :

- *(1) schéma explicite (ou progressif, "forward" en anglais) :*

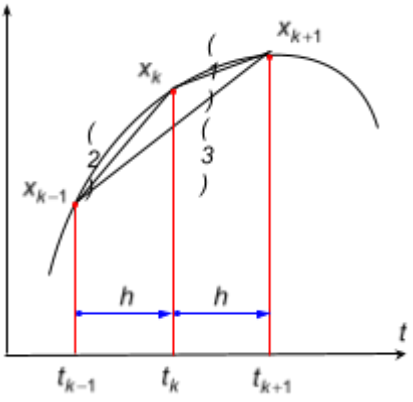
$$\frac{dx(t_k)}{dt} \approx \frac{x_{k+1}-x_k}{t_{k+1}-t_k} \approx \frac{x_{k+1}-x_k}{h} \quad ,$$

- *(2) schéma implicite (ou rétrograde, "backward" en anglais) :*

$$\frac{dx(t_k)}{dt} \approx \frac{x_k-x_{k-1}}{t_k-t_{k-1}} \approx \frac{x_k-x_{k-1}}{h}$$

- *(3) schéma centrée (schéma de Crank-Nicolson) :*

$$\frac{dx(t_k)}{dt} \approx \frac{x_{k+1}-x_{k-1}}{t_{k+1}-t_{k-1}} \approx \frac{x_{k+1}-x_{k-1}}{2h}$$



Seul le schéma explicite est officiellement au programme de 1ière année !

B – Rappel du principe de la méthode d'Euler, mise en œuvre avec un schéma explicite.

Une équation différentielle d'ordre 1 est une équation qui peut se mettre sous la forme : $\frac{dy(t)}{dt} = f\left(y(t),t\right)$.

On note :

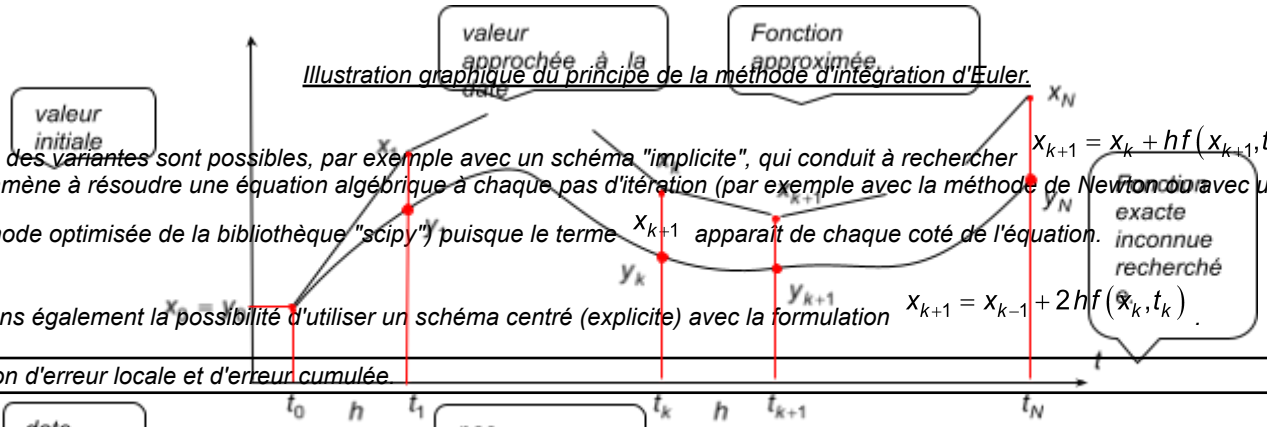
- $y(t)$ *la fonction solution exacte (inconnue) de l'équation différentielle donnée (ordre 1, linéaire ou pas),*
- $(t_0,t_1,...,t_k,t_{k+1},...,t_{N-1},t_N)$ *l'ensemble des temps pour lesquels on cherche une valeur approchée de* $y(t)$,
- $y_k = y\left(t_k\right)$ *les valeurs de la fonction exacte (inconnue !) pour les dates choisies,*
- $x_k = x\left(t_k\right)$ *les valeurs approchées construites par itération pour les dates choisies.*

La stratégie itérative mise en œuvre dans la méthode d'Euler consiste, à partir d'une condition initiale $x_0 = y_0 = y\left(t_0\right)$ *, à rechercher avec un schéma itératif (ici explicite) une valeur approchée* x_{k+1} *de la valeur exacte* $y_{k+1} = y\left(t_{k+1}\right)$ *avec la formulation suivante :*

$$x_{k+1} = x_k + \left(t_{k+1} - t_k\right)\frac{dy\left(t_k\right)}{dt}$$,

Si on travaille avec un pas d'intégration constant, soit : $h = t_{k+1} - t_k = cste$ *, et que l'on approxime la dérivée avec un schéma explicite, c'est-à-dire tel que* $\frac{dy(t_k)}{dt} \approx f\left(x_k,t_k\right)$ *, alors, il vient :* $x_{k+1} = x_k + hf\left(x_k,t_k\right)$ *, pour* $k \in [0,N-1]$.

Illustration graphique du principe de la méthode d'intégration d'Euler.



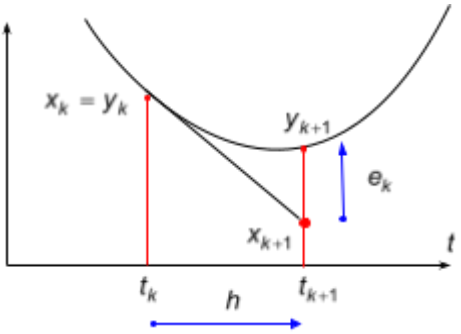
Mais des variantes sont possibles, par exemple avec un schéma "implicite", qui conduit à rechercher $x_{k+1} = x_k + hf\left(x_{k+1},t_{k+1}\right)$ *, ce qui amène à résoudre une équation algébrique à chaque pas d'itération (par exemple avec la méthode de Newton ou avec une méthode optimisée de la bibliothèque "scipy") puisque le terme* x_{k+1} *apparaît de chaque coté de l'équation.*

Notons également la possibilité d'utiliser un schéma centré (explicite) avec la formulation $x_{k+1} = x_{k-1} + 2hf\left(x_k,t_k\right)$.

Notion d'erreur locale et d'erreur cumulée.

A chaque itération, on commet une erreur e_k *dite "erreur de consistance" définie comme l'écart* $y_{k+1} - x_{k+1}$ *en supposant que le point de départ de l'itération* k *était* $x_k = y_k$ *, l'utilisation des développements limités à l'ordre 2 de Taylor-Lagrange permet de montrer que :*

$$e_k = y_{k+1} - x_{k+1} = y_{k+1} - \left(y_k + h\,y'\left(t_k\right)\right) = \boxed{\frac{h^2}{2}y''\left(t_k\right)} + \mathbf{O}(h^2)$$.



On vérifie que l'ordre de grandeur de l'erreur cumulée : $|e_0| + |e_1| + ... + |e_{N-1}|$

est celui de l'erreur globale ε_n *, définie comme* $\varepsilon_n = \max_{0 \le j \le N} |y_j - x_j|$.

C – Mise en œuvre sur un exemple simple.

Dans un premier temps, afin de vérifier la pertinence de la méthode d'Euler, on propose de l'appliquer à une équation différentielle simple intégrable de manière à pouvoir comparer le résultat obtenu avec la solution analytique.

On considère ainsi l'équation différentielle : $\frac{dy(t)}{dt} = \omega \sin(\omega t)$, avec $\omega \in \mathbb{R}^{*+}$ et avec une condition initiale de la forme : $y(0) = y_0$.

Le fichier `IPT-TD3-IntNum-C_vEleve.py` *comporte un script détaillé (mais incomplet !) de la mise en œuvre de la méthode d'Euler à l'équation différentielle proposée (avec tracés comparatifs des solutions).*

Il doit être préalablement placé dans votre espace personnel de travail avant de poursuivre.
Toute les zones entre triple quote (' ') sont à compléter.

Programmation partielle sous Python.

1. Compléter le script proposé pour intégration numérique avec Euler et "scipy.integrate.odeint" en définissant :

(a) - la fonction **`deriv_eq(y,t)`**, de manière à renvoyer la valeur de la dérivée de la fonction à la date (t),
(b) - la fonction **`euler_explicite(y,t)`**, de manière à correspondre à la méthode d'Euler (*schéma explicite*),
(c) - la fonction **`solution_exacte(t)`**, qui soit solution analytique exacte de l'équation différentielle.

2. Saisir les valeurs suivantes : $\omega = 5[\text{rad}\cdot\text{s}^{-1}]$, $y(0) = 0$, t_0=0 , t_1=10 , $n = 50$.
Compiler votre script et observer les résultats et comparer leurs performances en temps CPU (!).

Que constate-t-on à propos de l'approximation obtenue avec la méthode optimisée "odeint" de la bibliothèque "scipy".

3. Augmenter le nombre de pas d'intégration, par exemple $n = 500$ et recompiler.
Constater la pertinence des deux méthodes et comparer leurs performances en temps CPU.

4. Augmenter encore le nombre de pas d'intégration, par exemple $n = 5000$ et recompiler.
Constater la pertinence des deux méthodes et comparer leurs performances en temps CPU.

Influence du pas d'intégration.

5. Saisir les valeurs suivantes : $\omega = 8[\text{rad}\cdot\text{s}^{-1}]$, $y(0) = 0$, t_0=0 , t_1=10 , $n = 20$.

Compiler votre script et observer la dérive des résultats de la méthode non optimisée (Euler).

Diminuer encore le nombre de pas, par exemple : $n = 15$ et recompiler.

6. Saisir la valeur particulière : : $n = 12$.

Recompiler et constater (!).

Conclusion :

La méthode d'Euler ainsi construite n'est pertinente que pour des pas d'intégration suffisamment "court" (par rapport aux fluctuations de la fonction recherchée qui, a priori, est inconnue !), la tentation est alors grande de choisir un très grand nombre de pas d'intégration, mais cela a une influence directe sur le temps de calcul (!).

La méthode optimisée de la bibliothèque "scipy.integrate" est particulièrement performante puisqu'elle "semble" insensible à la discrétisation choisie (!). Pour comprendre l'optimisation qui a été programmé dans cette fonction, il faudrait "ouvrir" le code source de cette bibliothèque (disponible puisque écrit en open-source !), mais cela demande un peu de temps et de pratique …

D - Problème proposé : *Résolution approchée d'une équation différentielle d'ordre 1 non linéaire.*
Performances 0-100km/h d'un véhicule typé "sport" Audi A6 3.0L V6-TD Multitronic.
(d'après Philippe FICHOU, Chateaubriand, Rennes)

Ce problème consiste à simuler par intégration numérique (de façon simplifiée mais réaliste !) les performances d'accélération d'un véhicule typé "sport" à partir de ses caractéristiques mécaniques (puissance, rendement, transmission, adhérence, aérodynamisme)

La transmission Multitronic
plupart des boites de vite
engrènent). La variation en
délivre la puissance maxima

Le constructeur Audi annonc
La vitesse maximale annonc

Les caractéristiques mécani

- $P_{max} = 150\ \text{kW}\ (204\ CV)$ *, puissance maximale du moteur,*
- $\eta = 0.9$ *, rendement global de la transmission mécanique,*
- $S = 2{,}35\ \text{m}^2$ *, surface frontale du véhicule (maître-couple),*
- $C_x = 0{,}3$ *, coefficient de pénétration dans l'air,*
- $k = 4\ \text{N}\cdot\text{s}\cdot\text{m}^{-1}$ *, coef. frottement des pneus (résistance au roulement),*

Les autres données sont :
- $f = 0{,}50$ *, coefficient d'adhérence maxi (sol/pneu) en conditions normales,*
- $\rho = 1{,}3\ \text{kg}\cdot\text{m}^{-3}$ *, masse volumique de l'air en conditions normales.*

L'équation différentielle qui caractérise l'évolution de la vitesse $v(t)$ *du véhicule est issue de l'écriture du principe fondamentale de la dynamique, avec les notations précédentes, et si l'on tient compte de la double limitation (limite d'adhérence des pneus sur le sol et limite de la puissance du moteur) on montre qu'elle peut s'écrire sous la forme compacte ci-dessous :*

$$M\frac{dv}{dt} = \underbrace{\min\left(T, \frac{P_{max}\,\eta}{v}\right)}_{F_m} - \underbrace{\frac{1}{2}\rho S C_x v^2}_{F_a} - \underbrace{kv}_{F_r},$$

(F_m, F_a, F_r) *étant des efforts, respectivement de motricité, de résistance aérodynamique et de résistance au roulement.*

Cette écriture compacte permet de décrire avec une unique équation l'évolution de l'accélération en utilisant la pleine puissance du véhicule et en tenant compte des limites d'adhérence, ce qui permet, par exemple, de décrire un départ arrêté (0 – 100 km/h), dans lequel cohabitent en réalité deux phases distinctes :

- une première phase où la limitation est une limitation en "adhérence" des pneumatiques sur le sol,
(au démarrage, toute la puissance ne peut pas être transmise aux roues du fait d'une adhérence limitée)

- une seconde phase où la limitation est une limitation en puissance du moteur, (au delà d'une certaine vitesse,
le moteur n'est plus capable de produire une force de traction supérieure à la limite d'adhérence)

Si, de plus, on considère que le véhicule est équipé d'une transmission de type "quattro" avec répartiteur électronique de puissance entre les essieux avant et arrière (ce qui est le cas de ces modèles haut de gamme), alors on montre que l'adhérence limite n'est plus fonction de l'accélération, mais uniquement des paramètres (f, M, g) *, on a alors :* $T \approx f\,M\,g$ *, adhérence maximale possible du véhicule (c'est-à-dire composante tangentielle de l'action mécanique sol/pneu) où* $g = 9{,}81\ \text{m}\cdot\text{s}^{-2}$ *est l'accélération de la gravité.*

Ouvrir le fichier `IPT-TD3-IntNum-D-AudiMultitronic_vEleve.py` .

Travail préliminaire : performances de motricité de la voiture, vitesse maximale théorique.

7. Compléter le script proposé de manière à estimer avec le script Python la valeur de la vitesse de transition v_t , (*vitesse pour laquelle la limite d'adhérence correspond à la puissance maximale du moteur*).
La faire afficher en $[\text{m/s}]$ et $[\text{km/h}]$. Vérifier qu'elle d'environ $50\,\text{km/h}$.

Rechercher également la vitesse maximale théorique de ce véhicule v_m , et la faire afficher en $[\text{m/s}]$ et $[\text{km/h}]$.

indication: dans un premier temps, on pourra négliger la résistance au roulement des pneumatiques de manière à résoudre une équation algébrique simple pour avoir une valeur (approchée) de v_m *, soit* `Vm1` *,*

puis dans un second temps on construira la fonction correspondant au calcul de la valeur exacte de v_m *, soit* `Vm2` *sans négliger le terme de résistance au roulement des pneumatiques et on confiera la recherche de cette racine à l'une des fonctions optimisées de recherche de racine de la bibliothèque "scipy.optimize":*

`scipy.optimize.fsolve ( f , x0 )`
`f` *étant la fonction dont on cherche un zéro,*
`x0` *étant une valeur de départ pour la recherche de la racine*

Définir une fonction $F_m(v)$ qui corresponde à $F_m = \min\left(T, \frac{P_{\max}\,\eta}{v}\right)$ (*qui traite correctement le cas particulier* $v = 0$ *!).*

Obtenir son tracé dans une figure avec en abscisse $v\,[\text{km/h}]$ et en ordonnée $F_m\,[\text{N}]$ pour $v \in [0, v_{\max}]$.

indication: il peut apparaître une erreur lors de l'exécution en raison de l'éventuelle présence d'un test dans votre définition de $F_m(v)$ *, interpréter cette erreur … (!)*

*pour contourner cette difficulté, le script propose de construire des listes (***`liste_v, liste_Fm, liste_Fr`***) et d'affecter les valeurs de l'une d'elle dans une boucle (pour éviter l'erreur avec le test).*

Constater que la cassure a bien lieu pour $v = v_t$.

Pour se rendre compte de la pertinence du tracé, compléter votre script avec le calcul de l'effort de résistance total (*aérodynamique et résistance au roulement des pneus*) que doit vaincre le véhicule, noté : $F_r\,[\text{N}]$ (**`Fr`** *sous Python*) et superposer sa courbe avec celle de $F_m\,[\text{N}]$. Que constate-t-on de prévisible ?

Faire constater au prof votre tracé... 

Travail spécifique : recherche de la courbe de vitesse par résolution approchée de l'équation différentielle avec la méthode d'Euler

8. Compléter le script proposé de la procédure d'intégration numérique avec la méthode d'Euler et un schéma explicite.

Cette procédure renvoie deux listes (`t , x`) où - `t` correspondant à la liste des temps avec un pas constant,
- `x` correspondant à la liste des valeurs approchées avec Euler.

Faire de même pour la procédure d'intégration numérique avec un schéma centré (*cf. rappel page 1*).

Choisir des conditions initiales qui permettent de simuler un départ arrêté avec une durée d'au moins 10 s et un faible nombre de pas d'intégration (*prendre 12 pour commencer*), compiler votre script.

Justifier le fait que les différentes méthodes donnent des valeurs approchées très semblables durant la première phase, puis divergent durant la seconde phase.

Vérifier que le fait d'augmenter le nombre de pas de calcul a un impact direct sur le temps de calcul mais diminue les différences observables entre les différentes méthodes.

Vérifier que le chrono annoncé par le constructeur est vraisemblable (!).

9. Faire une simulation sur 120 s avec 250 étapes, puis avec 2500 étapes et enfin 10 000 étapes. Constater.

E - Problème proposé : *Résolution approchée d'une équation différentielle d'ordre 1 (linéaire et non linéaire).*
Simulation comparée des performances de chauffage d'un bâtiment selon deux techniques différentes.
(sur une idée de Philippe FICHOU, Chateaubriand, Rennes)

Ce problème consiste à simuler par intégration numérique (de façon simplifiée mais réaliste !) les performances de deux modes de chauffage d'un bâtiment (chauffage électrique et pompe à chaleur) sans puis avec prise en compte des variations de températures au cours d'un cycle nuit/jour.

En toute rigueur, les températures apparaissant dans un modèle de thermodynamique sont exprimées en (K), par soucis de simplification dans l'interprétation et dans l'affichage des courbes, toutes les températures mentionnées ici seront en (°C) !

Les pertes énergétiques du bâtiment sont modélisées par des fuites thermiques avec le modèle de Newton, soit : $P_f = h_f\left(T - T_{ext}\right)$,

Le bâtiment a une capacité thermique $C\left[\mathrm{J\cdot K^{-1}}\right]$ *. L'équation de bilan de la chaleur conduit, quelle que soit l'installation, à l'équation :*

$\frac{dT}{dt} + \frac{h_f}{C}\left(T - T_{ext}\right) = \frac{P_p}{C}$ *, où* P_p *correspond à la puissance de chauffage produite dans le bâtiment.*

En pratique, la différence de performance entre ces deux modes de chauffage provient du fait qu'un chauffage électrique ne produit que ce qu'il consomme en énergie électrique, soit : $P_p = P_e$ *, tandis qu'une pompe à chaleur utilise une partie des calories du milieu extérieur pour produire une puissance :* $P_p = P_c \frac{T + 273}{\left(T - T_{ext}\right)}$ *, (*P_c *= puissance du compresseur), ainsi tant que* $T > T_{ext}$, $P_p > P_c$.

En résumé, pour une installation avec convecteurs électriques, l'équation de bilan est : $\frac{dT}{dt} + \frac{h_f}{C}\left(T - T_{ext}\right) = \frac{P_e}{C}$,

tandis que pour une installation avec pompe à chaleur, l'équation de bilan est : $\frac{dT}{dt} + \frac{h_f}{C}\left(T - T_{ext}\right) = \frac{P_c}{C}\frac{T + 273}{\left(T - T_{ext}\right)}$

Les caractéristiques de l'installation sont :

- $C = 10^7\ \mathrm{J\cdot K^{-1}}$ *, capacité thermique du bâtiment,*
- $h_f = 400\ \mathrm{W\cdot K^{-1}}$ *, coefficient de déperdition énergétique,*
- $P_e = 6\ \mathrm{kW}$ *, puissance électrique de l'installation "chauffage électrique",*
- $P_c = 0{,}4\ \mathrm{kW}$ *, puissance électrique du compresseur d'une "pompe à chaleur",*
- $T_0 = 10\ \mathrm{°C}$ *, température initiale interne du bâtiment,*
- $T_{ext0} = 3\ \mathrm{°C}$ *, température extérieure initiale (minimum durant la nuit),*
- $\Delta = 5\ \mathrm{°C}$ *,amplitude de variation de la température au cours d'un cycle (nuit/jour).*

Pour le mode électrique, si la température extérieure est constante $T_{ext} = cste$ *alors l'équation de bilan :* $\frac{dT}{dt} + \frac{h_f}{C}\left(T - T_{ext}\right) = \frac{P_e}{C}$

peut, avec le changement de variable $\theta = T - T_{ext}$ *, se réécrire :* $\frac{d\theta}{dt} + \frac{h_f}{C}\theta = \frac{P_e}{C}$ *, soit :* $\theta + \frac{C}{h_f}\frac{d\theta}{dt} = \frac{P_e}{h_f}$.

On reconnaît ainsi une équation différentielle d'ordre 1 linéaire dont la valeur finale est $\theta_\infty = \frac{P_e}{h_f}$ *et la constante de temps* $\tau = \frac{C}{h_f}$.

Ouvrir le fichier `IPT-TD3-IntNum-E-Chauffage_vEleve.py`.

Simulation des performances des deux modes de chauffage avec une hypothèse de température extérieure constante.

10. Compléter le script sous python par les définitions de :

(1) la fonction **`T_ext(t)`**, qui correspond, dans un premier temps, à une température constante ($(=T_{ext0})$,

indication : *définir* $T_{ext}(t)=T_{ext0}$ *est tentant mais provoquera un bug difficile à déceler, en effet, définir une fonction de (t) dans laquelle (t) n'apparaît pas ne pose pas de problème tant que l'on applique la fonction à un argument qui soit une unique valeur, mais l'appliquer à un argument de type liste ne produit pas une liste comportant autant de valeurs que la liste initiale, mais produit une liste ne comportant qu'une seule valeur !*

on "trichera" en définissant $T_{ext}(t)=T_{ext0}+0\times t$ *de manière à obtenir une liste…*

(2) la fonction **`f_ele(T,t)`**, de manière à renvoyer la dérivée $\frac{dT}{dt}$ avec le mode électrique,

(3) la fonction **`f_pac(T,t)`**, de manière à renvoyer la dérivée $\frac{dT}{dt}$ avec le mode pompe à chaleur.

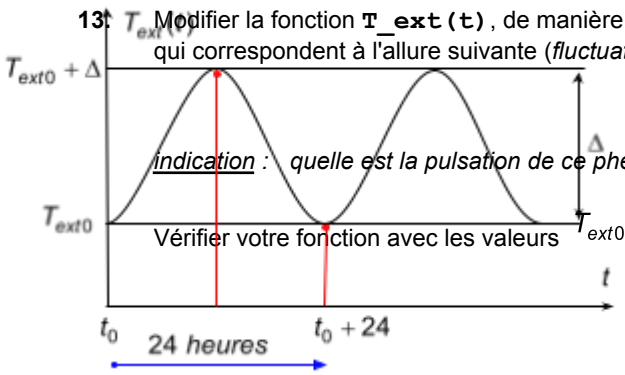
11. Compléter la fin du script de manière à obtenir les tracés souhaités.
indication : faire en sorte que l' échelle en abscisse soit graduée en (heure) et non en (s) !

12. Saisir les valeurs des paramètres pour l'intégration numérique de manière à simuler le fonctionnement sur 4 jours avec une température extérieure constante de 3°C et environ 100 pas pour l'intégration.

Constater l'intérêt d'une pompe à chaleur en terme de confort et de consommation énergétique par rapport à un convecteur électrique.

Simulation des performances des deux modes de chauffage avec une hypothèse de température extérieure fluctuante.

13. Modifier la fonction **`T_ext(t)`**, de manière à simuler des fluctuations de température extérieure (*cycle nuit/jour*) qui correspondent à l'allure suivante (*fluctuations sinusoïdales*) :



indication : quelle est la pulsation de ce phénomène ?

Vérifier votre fonction avec les valeurs $T_{ext0}=0\,°C$ et $\Delta=7\,°C$ et compiler.

pour aller encore plus loin :
Simulation de la dynamique du chauffage avec prise en compte d'un thermostat et d'une plage de température cible.

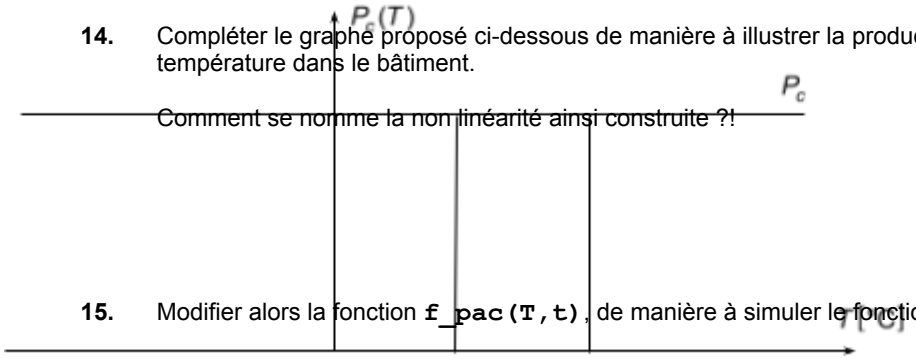
On souhaite simuler le fonctionnement de l'installation avec la présence d'un thermostat qui cible non pas une température unique mais une plage de température $\left[T_{basse}, T_{haute}\right]$. La stratégie mise en œuvre est telle que le thermostat :

- interrompt le chauffage lorsque la température devient supérieure à une température T_{haute} ,
- ne le réenclenchera que lorsque la température devient inférieure à une température T_{basse} .

Cette stratégie évite de trop solliciter l'installation par d'incessants cycles d'arrêts-redémarrages autour d'une valeur cible tout en maintenant la température dans une zone "cible". En effet, de trop fréquents arrêts et redémarrages de l'installation diminuent le rendement et la durée de vie des installations.

14. Compléter le graphe proposé ci-dessous de manière à illustrer la production de chauffage $P_c(T)$ en fonction de la température dans le bâtiment.

Comment se nomme la non linéarité ainsi construite ?!



15. Modifier alors la fonction `f_pac(T,t)`, de manière à simuler le fonctionnement de l'installation avec un thermostat.

<u>indication</u>: on se rends alors compte qu'il est utile de définir une variable intermédiaire "indic" (de type booléen) qui permettra de savoir si il faut ou non enclencher le chauffage lorsque la température se situe dans l'intervalle "cible" $\left[T_{basse}, T_{haute}\right]$.

C – Mise en œuvre sur un exemple simple : $\frac{dy(t)}{dt} = \omega \sin(\omega t)$, soit $y(t) = y_0 + 1 - \cos(\omega t)$

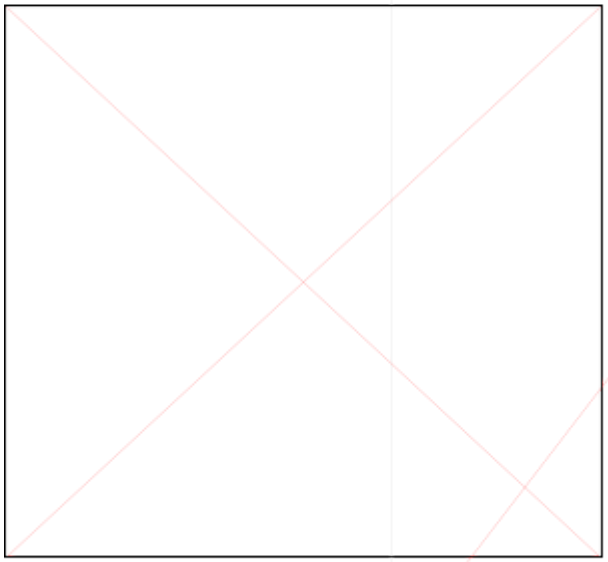
```python
# -*- coding: utf-8 -*-
#-------------------------------------------------------------------------------
# Intégration numérique avec la méthode d'Euler
# Exercice C : Application à une équation d'ordre 1
#-------------------------------------------------------------------------------

from math import *       # chargement des opérateurs mathématiques
import numpy as np       # pour utilisation des opérateurs sur des listes
from scipy.integrate import odeint   # pour comparaison avec Euler
import pylab as plt      # pour tracer des courbes proprement
import time              # pour évaluer le temps CPU consommé


# fonction qui renvoie la valeur de la dérivée dy/dt = f( y , t )
# pour correspondre à l'équation différentielle à intégrer numériquement
def deriv_eq(y,t):
    return omega*np.sin(omega*t)

# routine d'Euler explicite qui renvoie deux listes ( t , y )
"""
    on passe en arguments :
    f, la valeur de la dérivée de la fonction, soit y'(t)
    y0, la condition initiale,
    ( t0 , t1 ), les bornes en temps entre lesquelles intégrer
    nb, le nombre de pas d'intégration dans la méthode

    on construit :
    h, la valeur du pas d'intégration numérique
    ( t , fya ) les listes des temps et de la fonction y approximée
"""

def euler_explicite( f , y0 , t0 , t1 , nb):
    h   = (t1 -t0)/nb           # définition du pas d'intégration numérique
    t   = np.arange(t0 ,t1 ,h)  # construction de la liste des temps
    fya = np.zeros(nb)          # déclaration d'une liste (fya) et affection à 0
    fya[0]  = y0                # condition initiale
    for i in range(nb -1):
        fya[i+1] = fya[i] + h*deriv_eq(fya[i],t[i])
    return(t,fya)


# définition de la pulsation, de CI, des bornes en temps et du pas d'intégration
omega           = 7
CI              = 2
tmin , tmax , n = 0 , 9 , 10

pas             = (tmax-tmin)/n     # calcul du pas d'intégration
```

```python
# utilisation de la procédure d'Euler
"""
    on commence par mémoriser le temps machine avec time.perf_counter() avant
    de lancer la procédure de manière à estimer le temps CPU consommé
    par la procédure, et on l'affiche en (ms)

    on note ( t1 , y1 ) les listes renvoyées par
    la méthode d'Euler pour l'intégration numérique
    de l'équation différentielle
"""
tcpu0 = time.perf_counter()
t1 , y1 = euler_explicite( deriv_eq,CI,tmin,tmax,n)
tcpu1 = time.perf_counter()
print('Temps CPU (ms) - Euler(explicite) :', (tcpu1-tcpu0)*1e3)

# utilisation de la routine "odeint" de "scipy.integrate"
"""
    on commence par construire une liste de dates (t2)
    (identique à (t1) pour comparer les deux méthodes
    avec un même pas d'intégration !),
    puis on fait appel à scipy.integrate.odeint,
    Particularité: cette procédure renvoie un (y2)
    qui est une liste de liste [[],[],...[]] (!)
"""
t2 = np.arange(tmin , tmax, pas)

tcpu0 = time.perf_counter()
y2 = odeint(deriv_eq , CI , t2)
tcpu1 = time.perf_counter()
print('Temps CPU (ms) - Scipy(odeint) :', (tcpu1-tcpu0)*1e3)

# définition de la réponse analytique (solution exacte)
"""
    on définit la fonction solution analytique exacte
    puis on construit une liste des dates (t10)
    avec un pas de temps nettement plus fin (x 10)
    de manière à pouvoir aisément comparer les méthodes numériques
    au résultat analytique exact (référence)
"""

def solution_exacte(t):
    return CI+1-np.cos(omega*t)

t10 = np.arange(tmin , tmax, pas/10)
y10 = solution_exacte(t10)

# tracés des résultats sur une même figure
plt.figure(1)
plt.title (r"$Comparaisons\ des\ résultats$")
plt.xlabel(r"$t(s)$")
plt.ylabel(r"$(rad\cdot\ s^{-1})$")

"""
    selon le nombre de points à afficher pour les tracés (seuil choisi = 30),
    on utilise ou non un marqueur 'o-' de manière à ne pas surcharger la figure
"""
if n<30:
    plt.plot( t1 , y1      , 'o-b'  , label=r'$Approximation\ Euler$', lw=1.5)
    plt.plot( t2 , y2[:,0] , 'o-r'  , label=r'$Approximation\ Scipy$', lw=1)
    plt.plot( t10, y10     , '--g' , label=r'$Solution\ exacte$')
else:
    plt.plot( t1 , y1      , '-b'  , label=r'$Approximation\ Euler$', lw=1.5)
    plt.plot( t2 , y2[:,0] , '-r'  , label=r'$Approximation\ Scipy$', lw=1)
    plt.plot( t10, y10     , '--g' , label=r'$Solution\ exacte$')
```

```
plt.grid(True)
plt.legend(loc=4)
plt.show()
```

Résultats préliminaires.

```
adhérence maximale T = M *g *f [N]: 10840.050000000001

vitesse de transition pour l'adhérence maximale Vt = Pmax * eta / Tmax:
12.453817094939598 [m.s-1] 44.83374154178256 [km.h-1]

vitesse maximale approchée en négligeant la résistance au roulement
66.53912695461337 [m.s-1] 239.54085703660812 [km.h-1]

vitesse maximale théorique avec résolution complète
[ 63.75303643] [m.s-1] [ 229.51093114] [km.h-1]
```

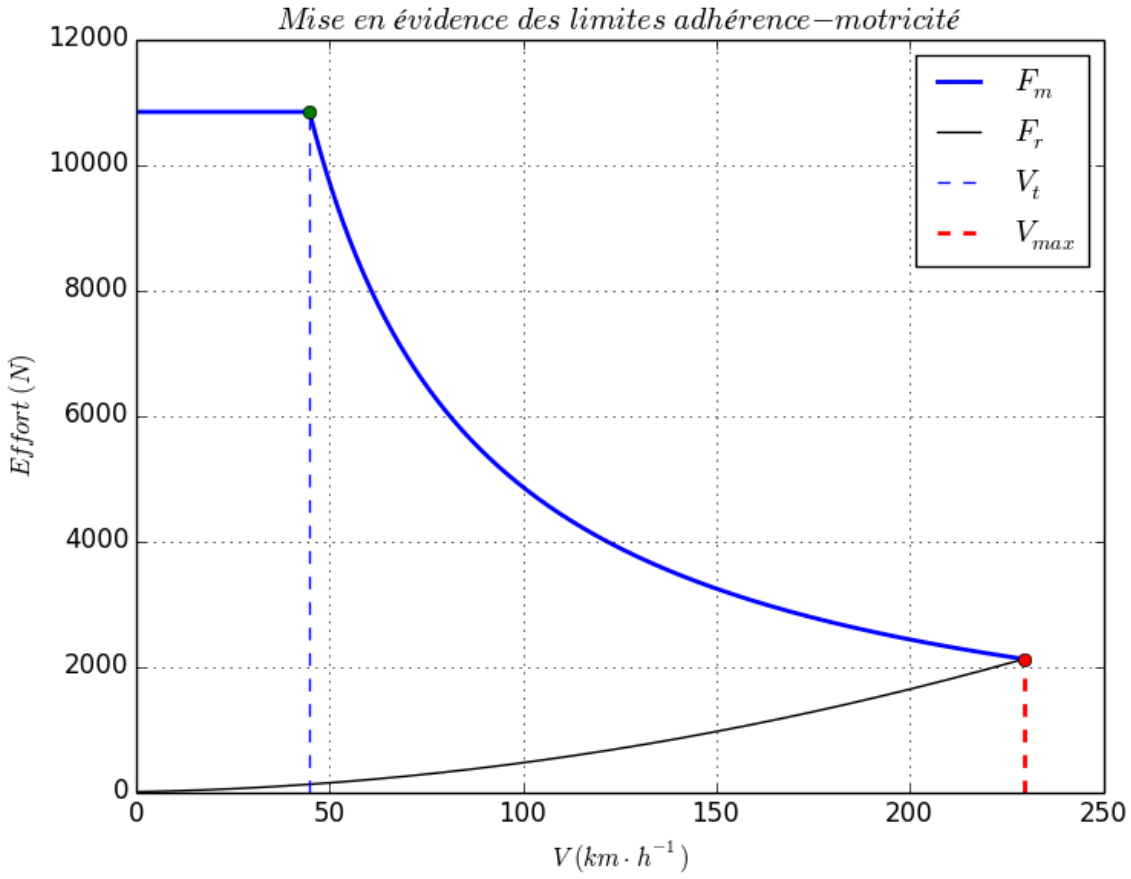


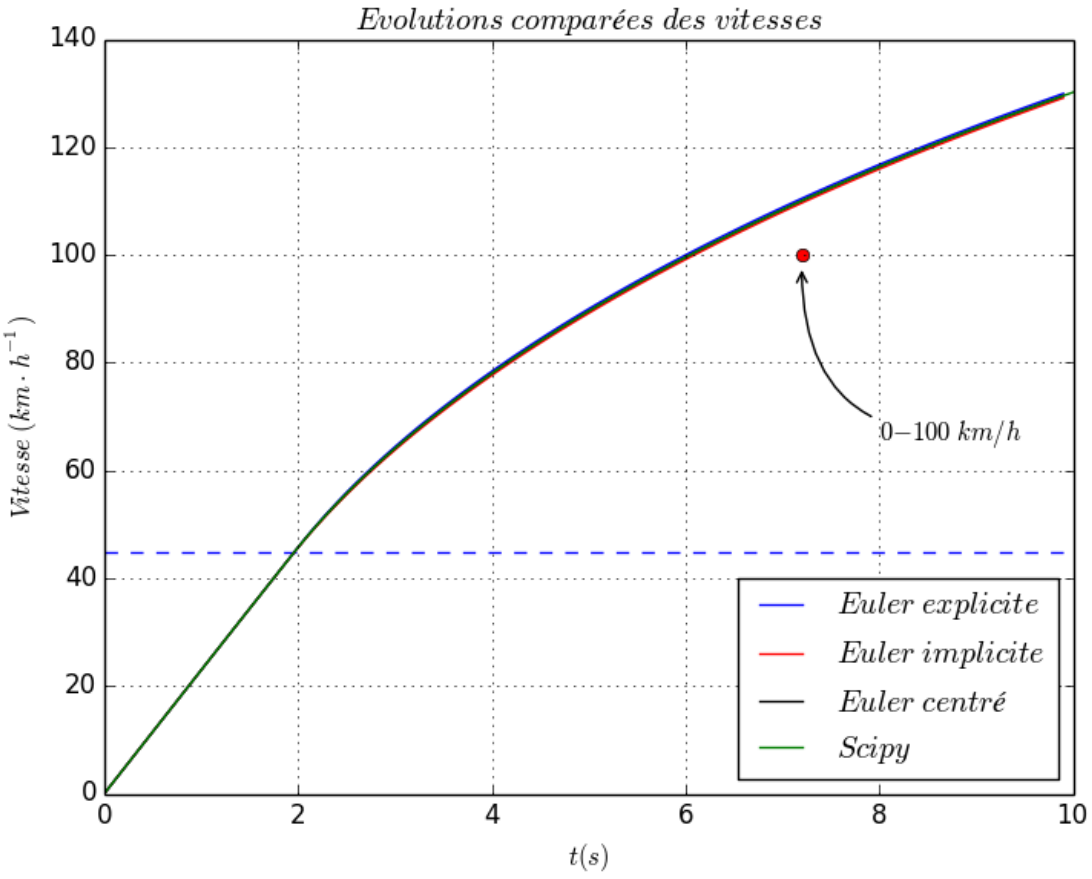
```python
# -*- coding: utf-8 -*-
#-------------------------------------------------------------------------------
# TP :   Intégration numérique / Equation différentielle non linéaire d'ordre 1
#        Méthode d'Euler / Application à la motricité d'une A6 Multitronic
#-------------------------------------------------------------------------------
from math import *  # pour pouvoir manipuler 'pi'
import pylab as plt
import numpy as np
import scipy.integrate
import scipy.optimize
import time

''' Travail préliminaire '''

# valeurs numériques
M     = 1700   # masse du véhicule (kg)
g     = 9.81   # accélération de la gravité (m.s-2)
f     = 0.65   # facteur d'adhérence pneu/route
rho   = 1.3    # masse volumique de l'air (kg.m-3)
S     = 2.35   # surface frontale = maitre-couple (m2)
Cx    = 0.3    # coeffcient de pénétration dans l'air
k     = 4      # facteur de résistance au roulement des pneus (N.s.m-1)
Pmax  = 150e3  # puissance maximale du moteur (W) 204cv = 150kW
eta   = 0.9    # rendement de la chaine de transmission
T     = f*M*g  # composante tangentielle maximale d'adhérence (Newton)

# calcul de la vitesse de transition Vt (adhérence maxi et puissance maxi)
Vt    = Pmax*eta/T

# fonction dont la racine est la vitesse maximale théorique
def f_vmax(v):
    return Pmax*eta/v-k*v-1/2*rho*S*Cx*v**2

# recherche de la vitesse maximale (1 = approchée, 2 = précise)
# arguments de fsolve ( fonction , valeur proche de la racine recherchée )
# les bornes doivent être telles que le produit f(a)*f(b) soit négatif !
Vm1  = (2*Pmax*eta/(rho*S*Cx))**(1/3)
Vm2  = scipy.optimize.fsolve(f_vmax,60)

print("adhérence maximale T = M *g *f [N]:",T)
print("vitesse de transition pour l'adhérence maximale Vt = Pmax * eta / Tmax:")
print(Vt,"[m.s-1]",Vt*3.6,"[km.h-1]")
print("vitesse maximale approchée en négligeant la résistance au roulement")
print(Vm1,"[m.s-1]",Vm1*3.6,"[km.h-1]")
print("vitesse maximale théorique avec résolution complète")
print(Vm2,"[m.s-1]",Vm2*3.6,"[km.h-1]")

# définition des fonctions Fm(v) et Fr(v)
def Fm(v):
    if v == 0:
        return T
    else:
        return min(Pmax*eta/v,T)

def Fr(v):
    return k*v+1/2*rho*S*Cx*v**2

# construction des listes de valeurs de Fm et Fr
# (Fr = résistance aéro + résistance au roulement)
# en fonction pour une liste de valeurs de la vitesse (v)
liste_v = np.linspace(0, Vm2, 500)
liste_Fr= Fr(liste_v)

liste_Fm= np.zeros(len(liste_v))
for j in range(len(liste_v)):
```

```
    liste_Fm[j] = Fm(liste_v[j])
```

```python
# synthèse et affichage des courbes Fm(v) et Fr(v)
plt.figure(0)
plt.title(r"$Mise\ en\ évidence\ des\ limites\ adhérence-motricité$")
plt.xlabel(r"$V\ (km\cdot\ h^{-1})$")
plt.ylabel(r"$Effort\ (N)$")
plt.plot(liste_v*3.6, liste_Fm, '-b' , linewidth = 2, label=r"$F_m$")
plt.plot(liste_v*3.6, liste_Fr, '-k' , linewidth = 1, label=r"$F_r$")
plt.plot([Vt*3.6,Vt*3.6],[0,T], '--b',                label=r"$V_t$" )
plt.plot([Vm2*3.6,Vm2*3.6],[0,Pmax*eta/Vm2],     '--r'     ,     linewidth     =     2,
label=r"$V_{max}$")
plt.legend(loc=1)
plt.plot(Vt *3.6,T       , 'og')      # pour marquer la cassure
plt.plot(Vm2*3.6,Fr(Vm2) , 'or')      # pour marquer la vitesse limite
plt.grid(True)
plt.show(0)


# équation différentielle de la motricité d'une Audi A6 avec boite Multitronic
# pour l'ensemble des phases (avec prise en compte de la double limitation)
# la fonction renvoie la valeur de la dérivée de la vitesse (dv/dt)
def fderiv(v,t):
    if v == 0:
        return 1/M*(T                  - k*v - 1/2*rho*S*Cx*v**2 )
    else:
        return 1/M*(min(Pmax*eta/v,T) - k*v - 1/2*rho*S*Cx*v**2 )


# définition des bornes pour l'intégration numérique
t0 , t1 , v0 , nb = 0 , 10 , 0 , 100


# procédure d'Euler avec schéma d'intégration explicite
# avec une équation différentielle sous la forme dx/dt = f (x(t),t)
# (t0,x0) = consition initiale, t1 = valeur finale, nb = nombre d'étapes

def euler_e(f, t0, t1, x0, nb):
    h = (t1 -t0)/nb
    t = np.arange(t0 ,t1 ,h)
    x = np.zeros(nb) # déclaration de la liste et affectation à 0
    x[0] = x0
    for i in range (nb -1):
        x[i+1] = x[i] + h*f(x[i],t[i])
    return (t,x)


# procédure d'Euler avec schéma d'intégration "implicite"
# avec une équation différentielle sous la forme dx/dt = f (x(t),t)
# (t0,x0) = condition initiale, t1 = valeur finale, nb = nombre d'étapes

def euler_i(f, t0, t1, x0, nb):
    h = (t1 -t0)/nb
    t = np.arange(t0 ,t1 ,h)
    x = np.zeros(nb) # déclaration de la liste et affectation à 0
    x[0] = x0
    for i in range (nb -1):
        def fff(u):
            return u-x[i]-h*fderiv(u,t[i+1])
        x[i+1] = scipy.optimize.fsolve( fff , x[i] )
    return (t,x)
```

```python
# procédure d'Euler avec schéma d'intégration centré
# avec une équation différentielle sous la forme x' = f (x(t),t)
# (t0,x0) = condition initiale, t1 = valeur finale, nb = nombre d'étapes

def euler_c(f, t0, t1, x0, nb):
    h = (t1 -t0)/nb
    t = np.arange(t0 ,t1 ,h)
    x = np.zeros(nb) # déclaration de la liste avant affectation de ses valeurs
    x[0] = x0
    x[1] = x0 + h*f(x[0],t[0]) # schéma explicite pour le second terme
    for i in range (1,nb-1):
        x[i+1] = x[i-1] + 2*h*f(x[i],t[i])
    return (t,x)


print(" ")
print("Travail spécifique:")
print("Intégration numérique avec la méthode d'Euler")

tcpu0 = time.perf_counter()
(te,ve) = euler_e(fderiv, t0,t1,v0,nb)
tcpu1 = time.perf_counter()
print("Temps CPU (en ms) avec Euler (explicite) :",(tcpu1-tcpu0)*1e3)

tcpu0 = time.perf_counter()
(ti,vi) = euler_i(fderiv, t0,t1,v0,nb)
tcpu1 = time.perf_counter()
print("Temps CPU (en ms) avec Euler (implicite) :",(tcpu1-tcpu0)*1e3)

tcpu0 = time.perf_counter()
(tc,vc) = euler_c(fderiv, t0,t1,v0,nb)
tcpu1 = time.perf_counter()
print("Temps CPU (en ms) avec Euler (centré)    :",(tcpu1-tcpu0)*1e3)


t = np.linspace(t0, t1, nb)
tcpu0 = time.perf_counter()
y = scipy.integrate.odeint(fderiv,v0,t)
tcpu1 = time.perf_counter()
print("Temps CPU (en ms) avec Scipy(odeint)    :",(tcpu1-tcpu0)*1e3)

plt.figure(1)
plt.title(r"$Evolutions\ comparées\ des\ vitesses$")
plt.xlabel(r"$t(s)$")
plt.ylabel(r"$Vitesse\ (km\cdot\ h^{-1})$")
plt.plot([t0,t1],[Vt*3.6,Vt*3.6], '--b' )#, label=r"$V_{adh}$")
plt.plot(te,ve*3.6, '-b' , label=r"$Euler\ explicite$")
plt.plot(ti,vi*3.6, '-r' , label=r"$Euler\ implicite$")
plt.plot(tc,vc*3.6, '-k' , label=r"$Euler\ centré$")
plt.plot(t ,y*3.6 , '-g' , label=r"$Scipy$")
plt.plot(7.2,100 , 'or' )
plt.annotate('$0-100\ km/h$', xy=(7.2,98), xytext=(8,70),va='top',
             arrowprops=dict(arrowstyle='->',
                             connectionstyle='angle3,angleA=0,angleB=90'))
plt.legend(loc=4)
plt.grid(True)
plt.show(1)
```


Mise en évidence des limites adhérence−motricité
F_m
F_r
V_t
V_{max}
$Effort\ (N)$
$V\,(km\cdot h^{-1})$
0
2000
4000
6000
8000
10000
12000
0
50
100
150
200
250


Evolutions comparées des vitesses
$0-100\ km/h$
Euler explicite
Euler implicite
Euler centré
Scipy
$Vitesse\ (km\cdot h^{-1})$
$t(s)$
0
20
40
60
80
100
120
140
0
2
4
6
8
10

```python
# -*- coding: utf-8 -*-
#-------------------------------------------------------------------------------
# TP :   Intégration numérique / Equation différentielle non linéaire d'ordre 1
#        Méthode d'Euler / Application à la comparaison de 2 modes de chauffage
#-------------------------------------------------------------------------------

from math import *  # pour pouvoir manipuler 'pi'
import pylab as plt
import numpy as np
import scipy as sp
import scipy.integrate
import scipy.optimize


# valeurs numériques
C     = 1e7    # capacité thermique du batiment (J.K-1)
hf    = 400    # coefficient de déperdition thermique (W.K-1)
Pe    = 6e3    # puissance électrique (W)
Pc    = 0.4e3  # puissance compresseur pompe à chaleur (W)
T0    = 10     # température initiale du local (°C)
Text0 = 0      # température extérieure initiale (°C)
Delta = 7      # variation de température (jour/nuit)

'''
    définition des variables pour la simulation du fonctionnement
    d'un thermostat dans une plage de température (Thaute, Tbasse)
'''
Thaute = 21    # température haute à laquelle se déclenche le thermostat
Tbasse = 19    # température basse à laquelle se ré-enclenche le thermostat
indic = 0      # initialisation indicateur ( = 1 si température haute atteinte)

# paramètres de l'intégration numérique
t0 , t1 , T0 , nb = 0 , 96*3600 , T0 , 1000


# procédure d'Euler avec schéma d'intégration explicite
# avec une équation différentielle sous la forme x' = f (x(t),t)
# (t0,x0) = condition initiale, t1 = valeur finale, nb = nombre d'étapes

def euler_e(f, t0, t1, x0, nb):
    h = (t1 -t0)/nb
    t = np.arange(t0 ,t1 ,h)
    x = np.zeros(nb)
    x[0] = x0
    for i in range (nb -1):
        x[i+1] = x[i] + h*f(x[i],t[i])
    return (t,x)

# procédure d'Euler avec schéma d'intégration "implicite"
# avec une équation différentielle sous la forme x' = f (x(t),t)
# (t0,x0) = condition initiale, t1 = valeur finale, nb = nombre d'étapes

def euler_i(f, t0, t1, x0, nb):
    h = (t1 -t0)/nb
    t = np.arange(t0 ,t1 ,h)
    x = np.zeros(nb)
    x[0] = x0
    for i in range (nb -1):
        def fff(u):
            return u-x[i]-h*f(u,t[i+1])
        x[i+1] = scipy.optimize.fsolve( fff , x[i] )
    return (t,x)
```

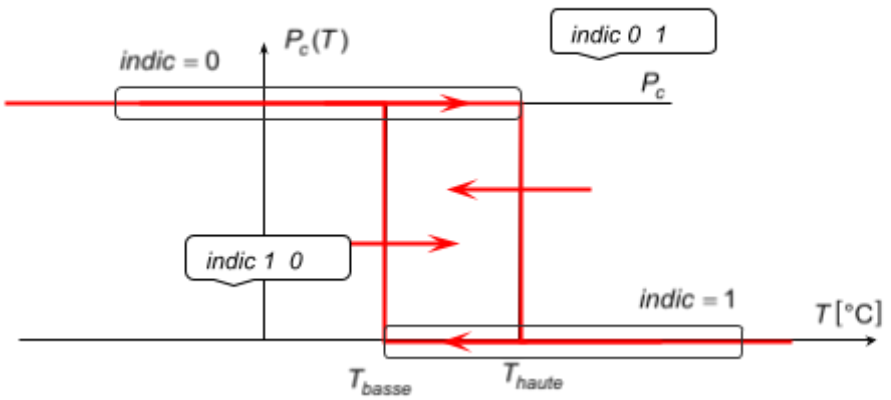
```python
# procédure d'Euler avec schéma d'intégration centré
# avec une équation différentielle sous la forme x' = f (x(t),t)
# (t0,x0) = condition initiale, t1 = valeur finale, nb = nombre d'étapes

def euler_c(f, t0, t1, x0, nb):
    h = (t1 -t0)/nb
    t = np.arange(t0 ,t1 ,h)
    x = np.zeros(nb)
    x[0] = x0
    x[1] = x0 + h*f(x[0],t[0]) # schéma explicite pour le second terme
    for i in range (1,nb-1):
        x[i+1] = x[i-1] + 2*h*f(x[i],t[i])
    return (t,x)

# fonctions utilisées pour les deux modèles de chauffage
# f_ele pour le modèle de chauffage électrique
# f_pap pour le modèle de pompe à chaleur
# ces fonctions renvoient la valeur de la dérivée dT/dt

def T_ext(t):
    return Text0 + Delta/2*(1-np.cos(t*2*pi/(24*3600)))

def f_ele(T,t):
    return 1/C * ( Pe - hf*(T-T_ext(t)) )

'''
    la fonction f_pac renvoie la valeur de la dérivée dT/dt
    et utilise un indicateur (indic) initialement = 0
    et qui permet de simuler le fonctionnement du thermostat
    si indic = 0, alors on désactive le chauffage lorsque T>Thaute
    (et dans ce cas on bascule l'indicateur à 1 pour mémoriser cette bascule)
    si indic = 1, alors on réenclenche le chauffage lorsque T<Tbasse
    (et dans ce cas on rebascule l'indicateur à 0)
'''
def f_pac(T,t):
    global indic, C, Pc, hf, Thaute, Tbasse

    if (indic == 0 and T <= Thaute) or (indic == 1 and T <= tbasse):
        indic = 0
        return 1/C * ( Pc*(T+273)/(T-T_ext(t)) - hf*(T-T_ext(t)) )

    elif (indic == 0 and T > Thaute) or (indic == 1 and T > Tbasse):
        indic = 1
        return 1/C * (                   - hf*(T-T_ext(t)) )

    else :
        print ("situation non envisagée ?!")
        return 0


# intégration numérique avec Euler pour différents schémas d'intégration
t , T_ele_e = euler_e(f_ele , t0 , t1 , T0 , nb)
t , T_ele_c = euler_c(f_ele , t0 , t1 , T0 , nb)
t , T_ele_i = euler_i(f_ele , t0 , t1 , T0 , nb)
t , T_pac_e = euler_e(f_pac , t0 , t1 , T0 , nb)
t , T_pac_c = euler_c(f_pac , t0 , t1 , T0 , nb)
t , T_pac_i = euler_i(f_pac , t0 , t1 , T0 , nb)
```

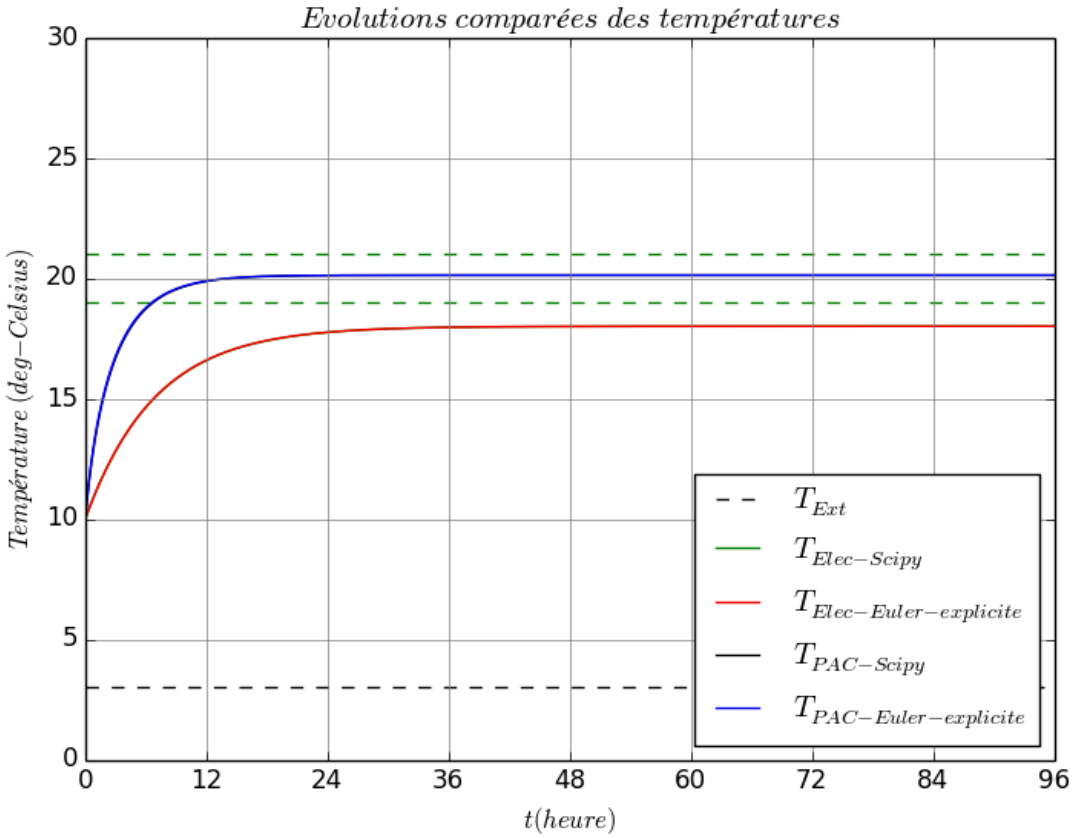
```python
# utilisation de la procédure d'intégation numérique odeint de scipy
# pour évaluer l'évolution de T en focntion du mode de chauffage
# (t) est une liste identique à celle construite avec Euler
# T_ele_scipy  et T_pac_scipy sont les listes des températures évaluées
# avec la procédure odeint de scipy.integrate
# respectivement en mode électrique et en mode pompe à chaleur

t = np.linspace(t0, t1, nb)
T_ele_scipy = sp.integrate.odeint(f_ele , T0 , t)
T_pap_scipy = sp.integrate.odeint(f_pac , T0 , t)


plt.figure()
plt.title(r"$Evolutions\ comparées\ des\ températures$")
plt.xlabel(r"$t(heure)$")
plt.ylabel(r"$Température\ (deg-Celsius)$")
plt.xticks(np.arange(t0/3600,t1/3600+12,12)) # grille par tranche de 12 heures
plt.ylim(0,30)                               # limitation de l'affichage à 30°C
plt.grid(color='grey',ls='-')
plt.grid(True)

plt.plot(t/3600 , T_ext(t)     , '--k' , label=r"$T_{Ext}$")
plt.plot(t/3600 , T_ele_scipy , '-g'  , label=r"$T_{Elec-Scipy}$")
plt.plot(t/3600 , T_ele_e     , '-r'  , label=r"$T_{Elec-Euler-explicite}$")
#plt.plot(t/3600 , T_ele_i     , '--r' , label=r"$T_{Elec-Euler-implicite}$")
#plt.plot(t/3600 , T_ele_c     , '--g' , label=r"$T_{Elec-Euler-centré}$")

plt.plot([t0/3600,t1/3600],[Thaute,Thaute], '--g')
plt.plot([t0/3600,t1/3600],[Tbasse,Tbasse], '--g')

plt.plot(t/3600 , T_pap_scipy , '-k'  , label=r"$T_{PAC-Scipy}$")
plt.plot(t/3600 , T_pac_e     , '-b'  , label=r"$T_{PAC-Euler-explicite}$")
#plt.plot(t/3600 , T_pac_i     , '--b' , label=r"$T_{PAC-Euler-implicite}$")
#plt.plot(t/3600 , T_pac_c     , '.-k' , label=r"$T_{Pompe-Euler-centré}$")

plt.legend(loc=4)
plt.show()
```

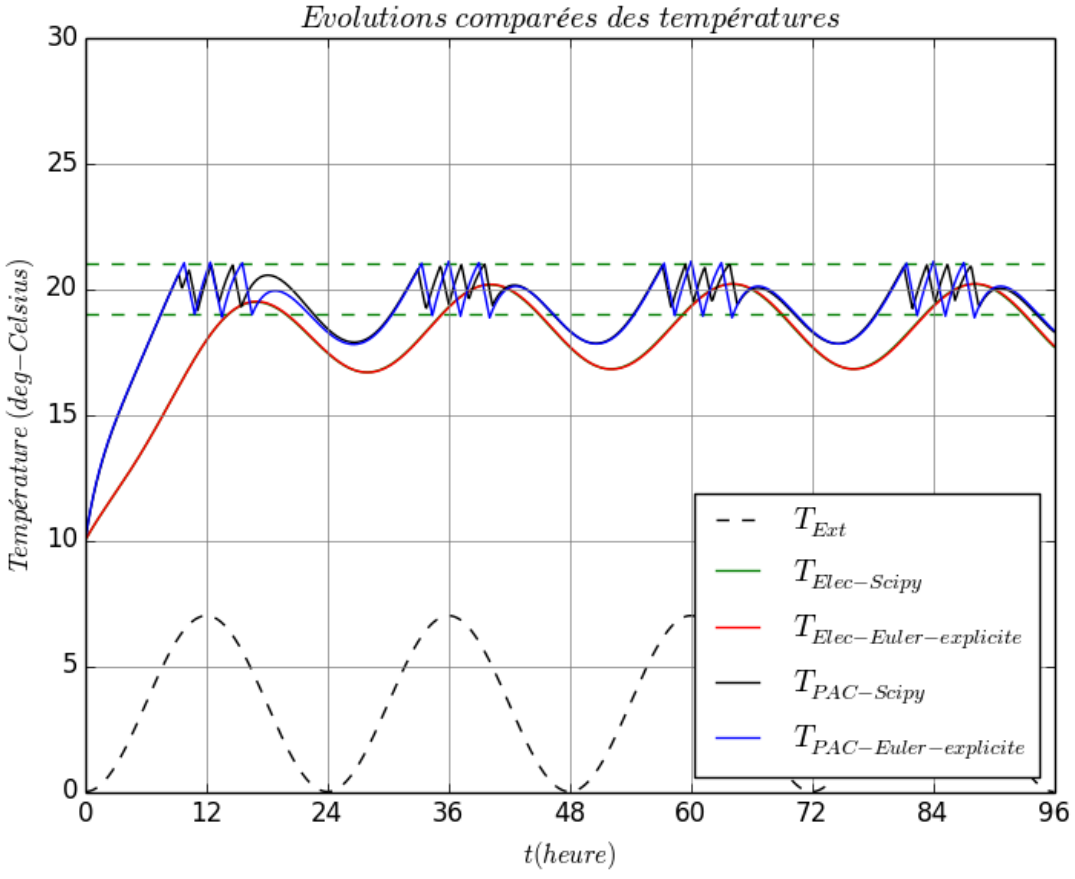
---

pour aller encore plus loin :
Simulation de la dynamique du chauffage avec prise en compte d'un thermostat et d'une plage de température cible.

Le fonctionnement du thermostat peut s'illustrer comme ci-dessous (*phénomène d'hystérésis*)




Evolutions comparées des températures
Température (deg−Celsius)
t(heure)
0
5
10
15
20
25
30
0
12
24
36
48
60
72
84
96
T_{Ext}
$T_{Elec-Scipy}$
$T_{Elec-Euler-explicite}$
$T_{PAC-Scipy}$
$T_{PAC-Euler-explicite}$


Evolutions comparées des températures
Température (deg−Celsius)
t(heure)
0
5
10
15
20
25
30
0
12
24
36
48
60
72
84
96
T_{Ext}
$T_{Elec-Scipy}$
$T_{Elec-Euler-explicite}$
$T_{PAC-Scipy}$
$T_{PAC-Euler-explicite}$